Autonomous Robot Controller Using Bitwise GIBBS Sampling

Rémi Canillas, Raphael Laurent, Marvin Faix, Dominique Vaufreydaz, Emmanuel Mazer

▶ **To cite this version:**

Rémi Canillas, Raphael Laurent, Marvin Faix, Dominique Vaufreydaz, Emmanuel Mazer. Autonomous Robot Controller Using Bitwise GIBBS Sampling. 15th IEEE International Conference on COGNITIVE INFORMATICS & COGNITIVE COMPUTING, Aug 2016, Calgary, Canada. hal-01316568

HAL Id: hal-01316568
https://inria.hal.science/hal-01316568

Submitted on 27 Aug 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Autonomous Robot Controller Using Bitwise GIBBS Sampling

Rémi CANILLAS[1], Raphaël LAURENT[2], Marvin FAIX[3], Dominique VAUFREYDAZ[4] and Emmanuel MAZER[5]

Author Version

Abstract

In the present paper we describe a bio-inspired non von Neumann controller for a simple sensorimotor robotic system. This controller uses a bitwise version of the Gibbs sampling algorithm to select commands so the robot can adapt its course of action and avoid perceived obstacles in the environment. The VHDL specification of the circuit implementation of this controller is based on stochastic computation to perform Bayesian inference at a low energy cost. We show that the proposed unconventional architecture allows to successfully carry out the obstacle avoidance task and to address scalability issues observed in previous works.

Keywords

Bayesian programming, stochastic computing, autonomous robots, Gibbs sampling.

1 Introduction

The present work is a part of the European BAMBI project (Bottom-Up Approaches to Machines dedicated to Bayesian Inference). The project's goal is to design low power unconventional (non von Neumann) bio-inspired probabilistic machines based on Bayesian inference and to use them as a tool to understand natural cognition. The main idea is that every living system processes information containing uncertainty, and at a microscopic level does so with very low energy consumption. Bayesian modeling, widely used in behavior prediction and decision making [1] [5], is used to emulate some of the qualities of these systems, on a macroscopic scale.

In previous works [3] [4], we have devised a compilation toolchain allowing to automatically generate VHDL circuit specifications of probabilistic machines computing inferences on Bayesian models using stochastic arithmetic. This first generation of Bayesian machines was based on the exhaustive inference paradigm: scanning all possible values of the discrete search space. While this approach lead to solving the problem of Pseudo Noise Sequence Acquisition [3] (a process by which emitters and receivers synchronize in law layers of telecommunication) and to control a simple sensorimotor robotic system [4] (which is able to make autonomous decisions by inferring proper motor commands from its sensor information), the complexity (in terms of number and cardinality of the variables) of the problems it can address remains limited. To address this scalability issue, we present in this paper a different approach based on approximate inference at the bit level. We show how a stochastic implementation of bitwise Gibbs sampling allows autonomous decision making in a simple sensorimotor robotic system, which we illustrate through an obstacle avoidance task.

This paper is organized as follows: we first give a quick overview of how our work fits in the landscape of research at the cross-point of probabilistic machines design and stochastic computing. We then specify the robotic problem of obstacle avoidance within the framework of Bayesian Programming (see [2]). We describe the stochastic architecture used to implement this solution without the need of a Floating Point Unit. Finally we show the results of experiments on a robot in a real life situation: a fully equipped modern apartment.

2 Previous Work

When evolving in uncontrolled environments, robots must handle uncertainty and need to take decisions based on incomplete data. Bayesian approaches provide an efficient way of addressing this challenge [8] [14] [2]. However, such approaches can be computationally expensive. This is why we propose to depart from the current paradigm: instead of running exact software computations on robust hardware, we choose to run stochastic computations

[1]CNRS, LIG - INSA Lyon, France, mail: remi.canillas@inria.fr
[2]ProbaYes SAS, Grenoble, France, mail: raphael.laurent@probayes.com
[3]Université Grenoble Alpes, CNRS, LIG, Grenoble, France mail: marvin.faix@inria.fr
[4]Université Grenoble Alpes, LIG, Inria, France, mail: dominique.vaufreydaz@inria.fr
[5]Université Grenoble Alpes, CNRS, LIG, France, mail: emmanuel.mazer@inria.fr

on dedicated hardware to dramatically reduce power needs (for instance [6] shows a decrease of several orders of magnitude).

Stochastic computing was introduced by the work of [16] and [7] as a way to offer low-cost highly parallel processing by coding probability values as temporal sequences and combining them with very simple operators. Because of a precision vs. computation time trade-off, stochastic computing has not been able to sustain the rise of faster processor units. Our approach keeps the simplicity of the stochastic operators on bitstreams, while getting rid of the computation time issue. Indeed, explicitly computing an accurate approximation of the target probability distribution is time expensive, and unnecessary when what you want is to choose one action relevant to your task: it suffices to select the first value given by the time sequence as a correct sample.

The idea of developing hardware dedicated to probabilistic computation is knowing a renewed interest among several teams [15] [10] [9] with goals similar to the BAMBI project's: exploring different computation paradigms to achieve better energy efficiency. Vigoda designed architectures [15] based on probabilities represented by analog signals, and used the message passing algorithm to compute exact inference. Mansinghka tries to resolve an approximate inference using sampling methods and Bayesian computation [10]. In a similar way, Jonas designed Markov Chain Monte Carlo based algorithms to provide a representation of probability distributions as sets of samplers [9]. Similar to the bio-inspired non von Neumann architectures of BAMBI Bayesian machines, the True North [12] project emulates a neuromorphic system for neuron networks, but still using fixed-point arithmetic units to compute a neuron's output knowing its inputs.

In [4], a robotic controller based on a probabilistic machine computing exact inference with approximate computation has been implemented. This exhaustive inference is very precise and fast on FPGAs, but presents a major drawback that we address in our present work: it requires a small amount of variables to be effectively implemented, which makes it not suitable for complex real-life applications. The aim of this paper is to show that (i) a stochastic architecture implementing bitwise Gibbs sampling allows to solve the obstacle avoidance task, and (ii) that this new implementation does not suffer from the previous scalability issues.

3 Bayesian controller for obstacle avoidance

In this section we present the Bayesian program allowing the robot to avoid obstacles. To allow for comparisons, we used the same robotic platform as described in [4], which provides a realistic set-up.

3.1 Variables

The small robot we used as a test platform is a simple sensorimoter system, perceiving its environment through three infrared sensors (IR_0, IR_1 and IR_2) coupled with three ultrasound sensors (US_0, US_1 and US_2). These sensors allow the robot to infer the distances (D_0, D_1 and D_2) to potential obstacles in three directions (D_0 being "front left", D_1 "front" and D_2 "front right") as described in Figure 1.

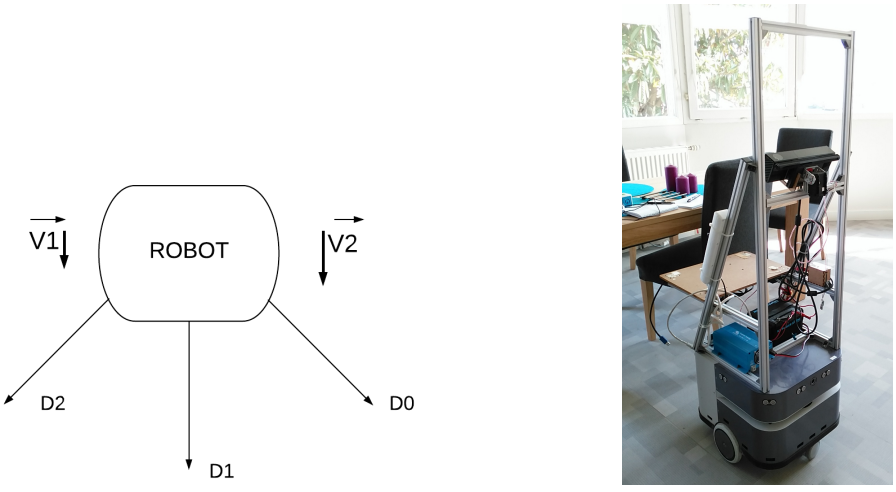


Figure 1: Schematic view and picture of the robot used for the experiment.

The robot, which is moving forward at a constant linear velocity, can adapt its trajectory by selecting as motor orders desired values of the differential rotation velocity, $V_{ROT} = V_2 - V_1$, which is the difference between the front left wheel rotational speed V_2 and the front right wheel rotational speed V_1, as shown in Figure 1.

3.2 Joint distribution decomposition

The joint probability distribution over the model variables $P(D_0D_1D_2IR_0IR_1IR_2US_0US_1US_2V_{ROT})$ is defined as a product of simpler probability distributions (priors, sensor models and action policy):

$$\begin{aligned} P(D_0D_1D_2IR_0IR_1IR_2US_0US_1US_2V_{ROT}) = \\ & P(D_0) \times P(D_1) \times P(D_2) \times \\ & P(IR_0|D_0) \times P(IR_1|D_1) \times P(IR_2|D_2) \times \\ & P(US_0|D_0) \times P(US_1|D_1) \times P(US_2|D_2) \times \\ & P(V_{ROT}|D_0D_1D_2) \ . \end{aligned} \tag{1}$$

3.3 Parametric forms

Each of the sensor models $P(IR_0|D_0)$, $P(IR_1|D_1)$, $P(IR_2|D_2)$, $P(US_0|D_0)$, $P(US_1|D_1)$ and $P(US_2|D_2)$ represents the confidence in the sensor reading given the actual distance of the object. They are modeled by Gaussian probability distributions whose mean is the sensor value and whose variance depends on the sensor's precision. The action policy $P(V_{ROT}|D_0D_1D_2)$ is modeled as a set of Gaussian probability distributions, which share the same variance, but whose mean value is computed as a function of the values of D_0, D_1 and D_2 with a simple model based on comparisons.

3.4 Probabilistic inference

To turn away from obstacles, the robot controller needs to select a motor command V_{ROT} knowing the values of the sensor readings. Bayesian inference yields:

$$\begin{aligned} & P(V_{ROT}|IR_0IR_1IR_2US_0US_1US_2) = \\ & \sum_{D_2} \Big[P(D_2)P(US_2|D_2)P(IR_2|D_2) \\ & \sum_{D_1} \Big(P(D_1)P(US_1|D_1)P(IR_1|D_1) \\ & \sum_{D_0} \left[P(D_0)P(US_0|D_0)P(IR_0|D_0)P(V_{ROT}|D_0D_1D_2) \right] \Big) \Big] \end{aligned} \tag{2}$$

In order to issue an order we simply draw a sample from this distribution on V_{ROT} and send it to the robot.

4 Stochastic architecture based on Gibbs sampling

To draw a sample from the target probability distribution $P(V_{ROT}|IR_0IR_1IR_2US_0US_1US_2)$, instead of using floating point computations, we propose to use a new type of architecture based on stochastic bitstreams which implements bitwise sampling of all unknown variables.

4.1 The Gibbs sampling algorithm for obstacle avoidance

Our system is composed of a set of 10 variables: $V = \{V_{ROT}, D_0, D_1, D_2, IR_0, IR_1, IR_2, US_0, US_1, US_2\}$. We partition V into three subsets: one subset containing our Known variables, $K = \{IR_0, IR_1, IR_2, US_0, US_1, US_2\}$; one subset containing the Searched variable, $S = \{V_{ROT}\}$; and finally one subset containing the Free variables: $F = \{D_0, D_1, D_2\}$. We can also define the subset containing the "Unknown" variables: $U = F \cup S = \{D_0, D_1, D_2, V_{ROT}\}$. We are interested in the probability distribution over the possible values of V_{ROT}, knowing the values of the six sensor readings IR_0, IR_1, IR_2, US_0, US_1, US_2. We use the Gibbs sampling algorithm as shown in Figure 2.

The main difficulty here is to compute a sample from $P(U_i|k_i, ..., k_{[K]}, u_1, ..., u_{i-1}, u_{i+1}, u_{[U]})$ without using a floating point unit. We address this issue by sampling this term bit by bit. Indeed, since we use only discrete variables (without loss of generality) their values have an equivalent binary representation.

4.2 Adapting the Gibbs algorithm to sample at the bit level

We explain our bitwise Gibbs sampling algorithm through the following example. Let us suppose that at a given step the system state is the following: $V_{ROT} = v_{rot}$, $D_0 = d_0$, $D_1 = d_1$, $D_2 = d_2$, $US_0 = us_0$, $US_1 = us_1$, $US_2 = us_2$. To get a sample of D_0, we need to draw according to the probability distribution

Initialization:
for $K_i \in K$ **do**
  Set K_i to the sensor reading value k_i
end for
for $U_i \in U$ **do**
  Draw a value u_i at random
end for
Sampling:
while TRUE **do**
  for $U_i \in U$ **do**
    Draw a sample u_i according to
      $P(U_i|k_i, ..., k_{[K]}, u_1, ..., u_{i-1}, u_{i+1}, u_{[U]})$
    Update U_i with the value u_i
  end for
end while

Figure 2: Gibbs algorithm used to compute samples of V_{ROT}.

$P(D_0 \mid d_1, d_2, ir_0, ir_1, ir_2, us_0, us_1, us_2)$.We will draw the next value of D_0 bit by bit. Let us suppose in the current state $d_0 = 1$ and we are drawing for instance a new value for the third bit of d_0 (the principle will be the same for the other bits). The odds of this third bit to be a 1 are defined by the following probability ratio:

$$O(D_0 \mid d_1, d_2, ir_0, ir_1, ir_2, us_0, us_1, us_2) = \frac{P(D_0 = 5 \mid d_1 d_2 ir_0 ir_1 ir_2 us_0 us_1 us_2)}{P(D_0 = 1 \mid d_1 d_2 ir_0 ir_1 ir_2 us_0 us_1 us_2)} \tag{3}$$

According to the Bayes theorem, we have:

$$P(D_0 \mid d_1 d_2 ir_0 ir_1 ir_2 us_0 us_1 us_2) = \frac{P(D_0 d_1 d_2 ir_0 ir_1 ir_2 us_0 us_1 us_2)}{P(d_1 d_2 ir_0 ir_1 ir_2 us_0 us_1 us_2)} \tag{4}$$

If we combine equations (3) and (4), the normalization constant disappears, and by using the decomposition of the joint probability distribution given by equation (1) we get:

$$\begin{aligned} O(D_0 \mid d_1 d_2 ir_0 ir_1 ir_2 us_0 us_1 us_2) = & \\ & \frac{P([D_0 = 5])\ P(d_1)\ P(d_2)}{P([D_0 = 1])\ P(d_1)\ P(d_2)} \times \\ & \frac{P(ir_0 \mid [D_0 = 5])\ P(ir_1 \mid d_1)\ P(ir_2 \mid d_2)}{P(ir_0 \mid [D_0 = 1])\ P(ir_1 \mid d_1)\ P(ir_2 \mid d_2)} \times \\ & \frac{P(us_0 \mid [D_0 = 5])\ P(us_1 \mid d_1)\ P(us_2 \mid d_2)}{P(us_0 \mid [D_0 = 1])\ P(us_1 \mid d_1)\ P(us_2 \mid d_2)} \times \\ & \frac{P(v_{rot} \mid [D_0 = 5]\ d_1 d_2)}{P(v_{rot} \mid [D_0 = 1]\ d_1 d_2)} \end{aligned} \tag{5}$$

The model terms which are uninformative for the considered variable disappear:

$$\begin{aligned} O(D_0 \mid d_1, d_2, ir_0, ir_1, ir_2, us_0, us_1, us_2) = & \\ & \frac{P([D_0 = 5])}{P([D_0 = 1])} \times \frac{P(ir_0 \mid [D_0 = 5])}{P(ir_0 \mid [D_0 = 1])} \times \\ & \frac{P(us_0 \mid [D_0 = 5])}{P(us_0 \mid [D_0 = 1])} \times \frac{P(v_{rot} \mid [D_0 = 5]\ d_1 d_2)}{P(v_{rot} \mid [D_0 = 1]\ d_1 d_2)} \end{aligned} \tag{6}$$

Our bitwise Gibbs algorithm needs to be able to draw all the bits of all variables (although after a proper burn-in period we are interested only on getting one correct sample of V_{ROT}). The same steps going from equations (3) to (6) can be used to show that the odds of any bit of any variable of our model can be obtained as a product of probability ratios.

4.3 Stochastic implementation of bitwise sampling

We compute these products by first generating a bitstream corresponding to each probability ratio using a Conditional Distribution Element (CDE), and then using an Extended C-Elements (ECEs) to calculate the products, as described in Figure 3.

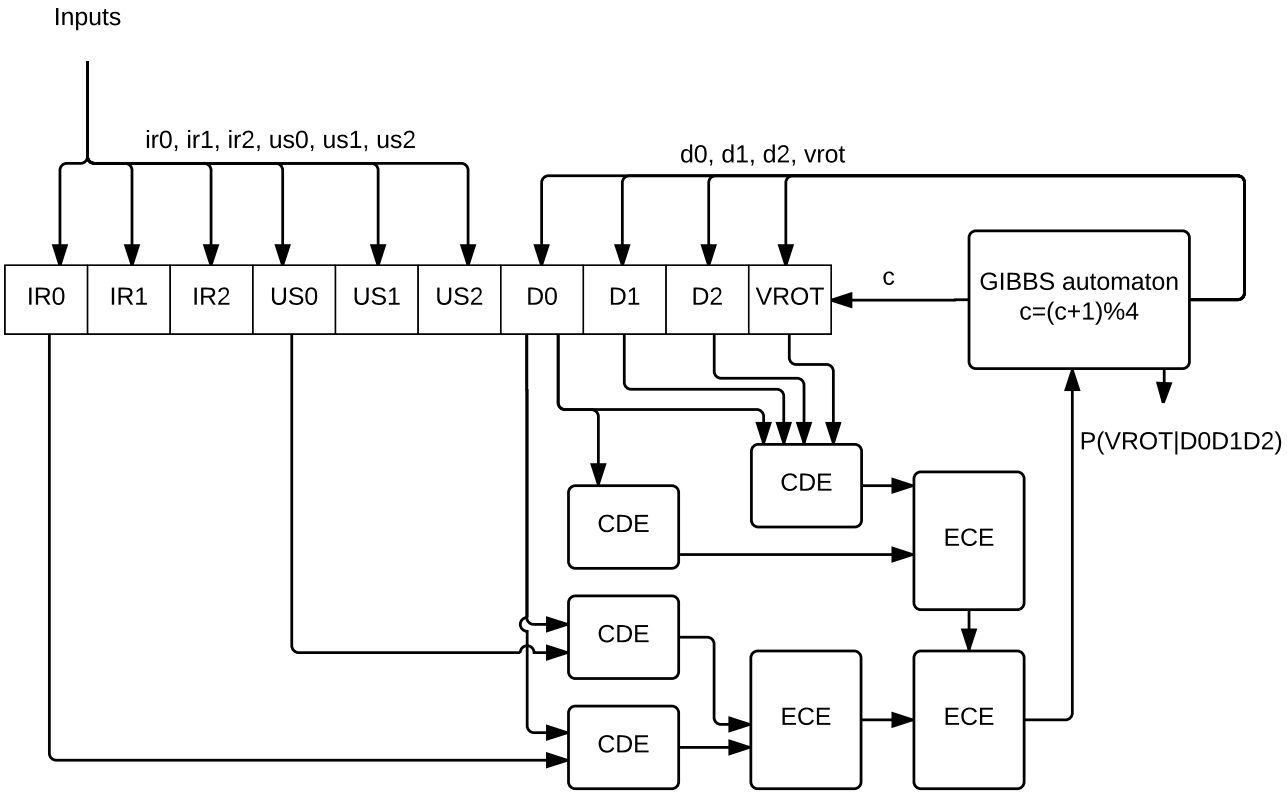


Figure 3: Architecture of a Gibbs sampler: the variable current values are stored in a bank of registers, and the Gibbs automaton chooses the c variable bit to update at each iteration while CDEs and ECEs compute the probabilistic operations needed. For clarity, we show only components needed to draw a new bit for D_0. The binarization of the variables is implicit.

In the architecture we propose, CDEs generate stochastic bitstreams which encode a probability ratio quantifying the likelihood of the considered bit to be equal to one given the value of the other bits. We define a stochastic bitstream as follows: in a stream of N bits, we note N_1 the number of bits of value 1. The bitstream encodes a value p_j when $\frac{N_1}{N} \to p_j$ when $N \to \infty$. The probability ratios we consider are homogeneous to some odds value o, from which we compute the bitstream probability $p = o/(1+o)$.

From the bitstreams encoding the probability ratios we compute a new bitstream the odds of which are the product of all probability ratios. For this purpose we use ECEs, which are an extension with more memory of Müller C-elements which have already been used to compute the "product in odds" operation in an energy efficient way to realize Bayesian inference [6]. Any bit of the resulting stream generated with the right probability can be used as the new sample.

4.4 Use of Gray code

Standard binary coding may induce some states to have very low probabilities, while they are mandatory intermediate states between two high probability states. This “probability desert” may prevent some high probability states from being reached in a reasonable time. This can be dealt with by using a Gray code [13] while interpreting our discrete variables as binary values, as shown Table 1. It guarantees that the coded values of any integer and its successor differ by exactly one bit.

Integer	0	1	2	3	4	5	6	7
Binary	000	001	010	011	100	101	110	111
Gray code	000	001	011	010	110	111	101	100

Table 1: Gray code for 3-bit integers

5 Experimentation

5.1 Controller implementation

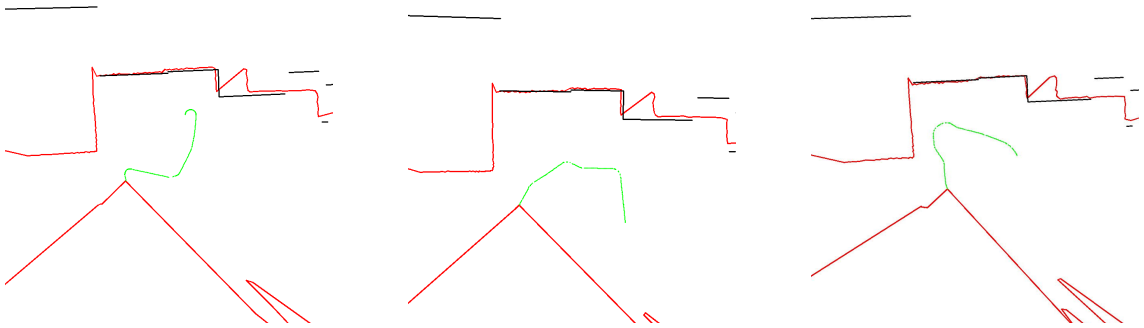
In order to implement the stochastic Gibbs circuit, a compilation toolchain was developed. It takes as input any Bayesian program on discrete variables used to specify the probability models, described with the ProBT [11] programming language. The outputs of this compilation toolchain are i) a VHDL program that precisely describes the architecture of the circuit used to perform stochastic inference using Gibbs sampling, and ii) a memory initialization file containing the odd ratios encoding the model specification. This compiler generated a circuit implementing the program from section 3.

This VHDL circuit specification will later be used for FPGA simulations and real circuit implementations, but for now we use it as an input for our own C++ simulation of the architecture, which is significantly faster in terms of computational speed than the standard EDA tool Modelsim. In our experiments we compare simulated architectures based on three different discretization of our values. The first one, named A, uses a discretization factor similar to the one used for the exhaustive inference controller in [4], namely a discretization on 3 values for distances and 5 for the rotation speed. The second one, named B, has 8 values for distances and 7 for the rotation speed (uneven number as one of them is "forward") and the third one, named C, has 16 values for distances and 15 for the rotation speed. One of our goals was to monitor the effect of a larger discretization in terms of computational speed and robot behavior. We restrain ourselves to these sets of values so as to be able to operate the robot in real time with our simulator, but of course future FPGA and circuit implementations will reduce computation time by several orders of magnitude.

5.2 Experimental setting

In order to check that our stochastic architectures successfully implement functional controllers, we tested them in a real life situation: we placed the robot in an apartment with obstacles such as chairs, tables and sofas. We then made it progress at a constant $0.15ms^{-1}$ linear velocity. The controller iterates (i) reading the sensor data, (ii) running the inference for 100ms and (iii) sending the inferred differential rotation velocity order so the robot can avoid potential obstacles. We made sure that the starting point was the same for the runs of the three different architectures. The picture in Figure 4 shows a view of the experimental ground where the obstacle avoidance took place.

Figure 4: View of the apartment used as our experiment field.



(a) Architecture A (b) Architecture B (c) Architecture C

Figure 5: Trajectories computed by the program embedded on our robotic platform. Ground map is represented by black lines. In red, one can see the view of obstacles detected by the robot laser sensor. Note that the robot is "blind" behind him with its lidar leading to the triangle at the bottom of each figure. In green, trajectories followed by the robot for three different architectures.

5.3 Results

5.3.1 Trajectories

Figure 5 shows a map of the apartment and the trajectories taken by the robot for our different architectures. We see that the robot is able to change its trajectory to avoid obstacles. A video of the experiment (using architecture B) is also available online [1]. We notice that on architecture A, the robot's trajectory angles are very abrupt, almost 90 degrees for each change of direction, and that for higher discretization values (architecture B and C) the trajectory is remarkably smoother.

5.3.2 Scalability of the proposed architecture

We compare in Table 2 the number of components for the three architectures A, B and C produced by the exhaustive inference controller described in [4], and our approximate inference controller. We see that the number of components exponentially increases when discretization is more precise for the exhaustive inference controller. We note that the number of components used by our approximate inference controller increases in a slower logarithmic way due to the bitwise sampling of our variables.

Architecture	A	B	C
Exhaustive inference architecture from [4]	1825	78190	2297310
Bitwise Gibbs sampling	92	119	149

Table 2: Comparison of the number of component instances in the VHDL circuits for two types of stochastic architectures.

6 Conclusion and Future Work

We presented a sensorimotor robot controller, defined by a Bayesian program, and implemented as an unconventional hardware architecture which does not require a Floating Point Unit, as shown in Figure 3. The obstacle avoidance task is realized by taking decisions through approximate inference thanks to Gibbs sampling at the level of stochastic bits. The proposed architecture relies on only two components: Conditional Distribution Elements (CDEs) to generate bitstreams sampling the probability quotients needed for Gibbs sampling, and Extended C-Elements (ECEs) to compute a bitstream encoding the product of these ratios.

We proved that this stochastic implementation of our robot controller was suitable for a real-life situation in a modern apartment, and this for 3 implementations corresponding to different discretizations of the problem's variables. While working with a relatively small amount of variables our controller is able to successfully replace a von Neumann processor in the decision process of this robotic system.

The fact that the scalability issue presented in [4] has been addressed in this paper allows us to believe that Bayesian systems based on stochastic architectures will be able to process more complicated applications such as occupancy filters and dynamic trajectory adjustment.

[1] https://www.youtube.com/watch?v=LcXwHg45jPM

Acknowledgment

This work was partly supported by the Agence Nationale de la Recherche (ANR) under the project Amiqual4Home ANR-11-EQPX-0002. It was made possible thanks to the EU collaborative FET Project BAMBI FP7-ICT-2013-C, project number 618024.

References

[1] P. Bessière, C. Laugier, and R. Siegwart. *Probabilistic Reasoning and Decision Making in Sensory-Motor Systems.* Springer, 2008.

[2] Pierre Bessière, Emmanuel Mazer, Juan Manuel Ahuactzin, and Kamel Mekhnacha. *Bayesian programming.* CRC Press, 2013.

[3] M. Faix, R. Laurent, J.Lobo, and E. Mazer. Cognitive computation: a bayesian machine case study. In *The 14th IEEE International Conference on Cognitive Informatics and Cognitive Computing.* IEEE, 2015.

[4] M. Faix, J. Lobo, R. Laurent, D. Vaufreydaz, and E. Mazer. Stochastic bayesian computation for autonomous robot sensorimotor systems. In *Workshop on Unconventional computing for Bayesian inference IROS.* IEEE, 2015.

[5] Joao Ferreira and Jorge Dias. *Probabilistic Approaches to Robotic Perception.* Spinger, 2013.

[6] Joseph Friedman, Lucie Calvet, Pierre Bessière, Jacques Droulez, and Damien Querlioz. Bayesian inference with Muller C-Elements. *IEEE Transactions on Circuits and Systems I*, 2016.

[7] BR Gaines. Stochastic computing systems. In *Advances in information systems science*, volume 2, pages 37–172. Springer, 1969.

[8] E. T. Jaynes. *Probability Theory: the Logic of Science.* Cambridge University Press, 2003.

[9] Eric Michael Jonas. *Stochastic architectures for probabilistic computation.* PhD thesis, Massachusetts Institute of Technology, 2014.

[10] Vikash Kumar Mansinghka. *Natively probabilistic computation.* PhD thesis, Massachusetts Institute of Technology, 2009.

[11] Kamel Mekhnacha, Juan-Manuel Ahuactzin, Pierre Bessière, Emanuel Mazer, and Linda Smail. Exact and approximate inference in ProBT. *Revue d'Intelligence Artificielle*, 21/3:295–332, 2007.

[12] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[13] Carla Savage. A survey of combinatorial gray codes. *SIAM Review*, 39(4):605–629, 1997.

[14] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics.* MIT Press, 2005.

[15] Benjamin Vigoda. *Analog logic: Continuous-Time analog circuits for statistical signal processing.* PhD thesis, Massachusetts Institute of Technology, 2003.

[16] John Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, 34:43–98, 1956.